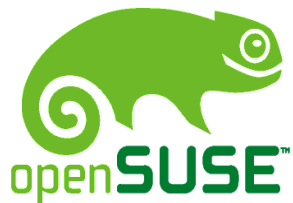


# Packages - The bricks we build with

---

Hendrik Vogelsang  
Packager  
Novell



Novell®

The purpose of this talk

Sources

---

project.c, project.h

configure, Makefile

man page, README

---

prepare the source code

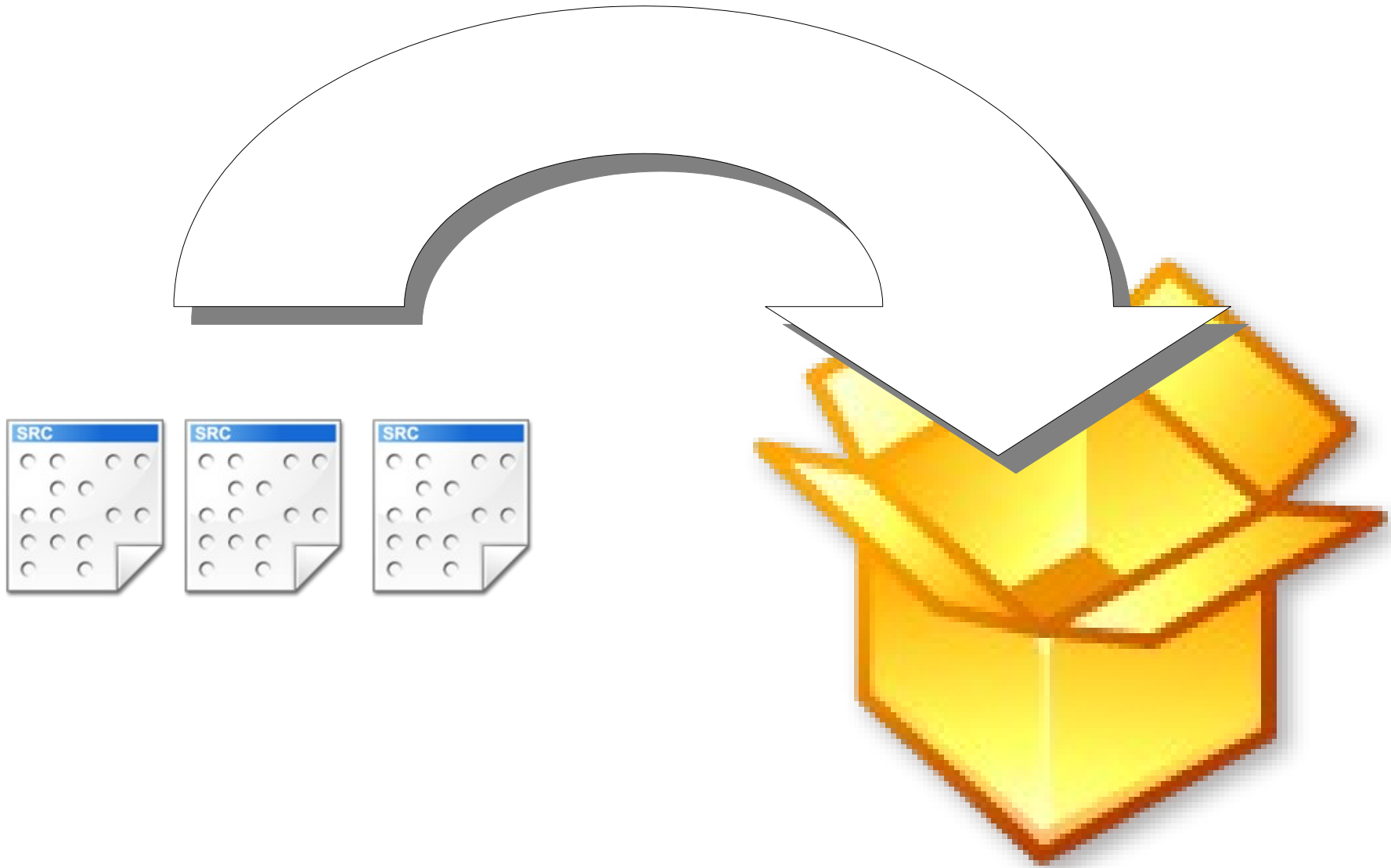
build executables

install executables  
and documentation

Packages

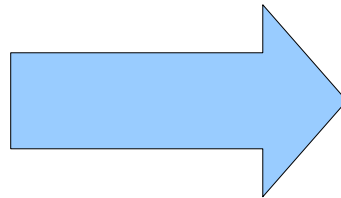


150.000





## Books and Magazines I



## Package Manifest



### Books and Magazines I

- Foundation by I. Asimov
- Cryptonomicon by N. Stephenson
- Wired (all of 2005)
- Time (all of 2005)

Packed 2006.03.01 by hvogel



### Books and Magazines II

- HHGG by Douglas Adams
- 1984 by George Orwell
- Wired (all of 2004)
- People (all of 2004)

Packed 2005.12.06 by hvogel

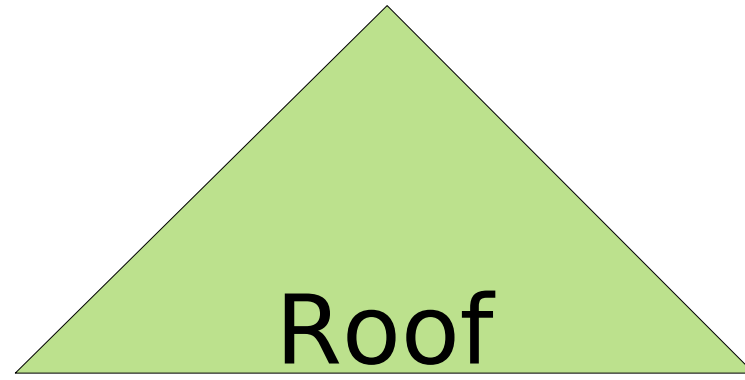


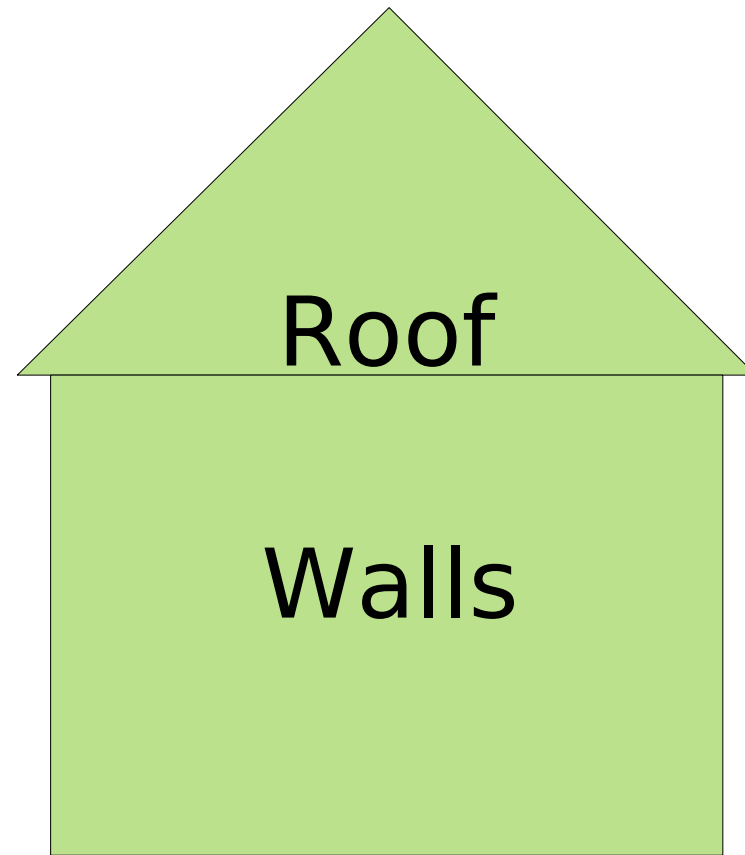
- Files (Box)
- Meta Data (Label)
- Package Database (Manifest)

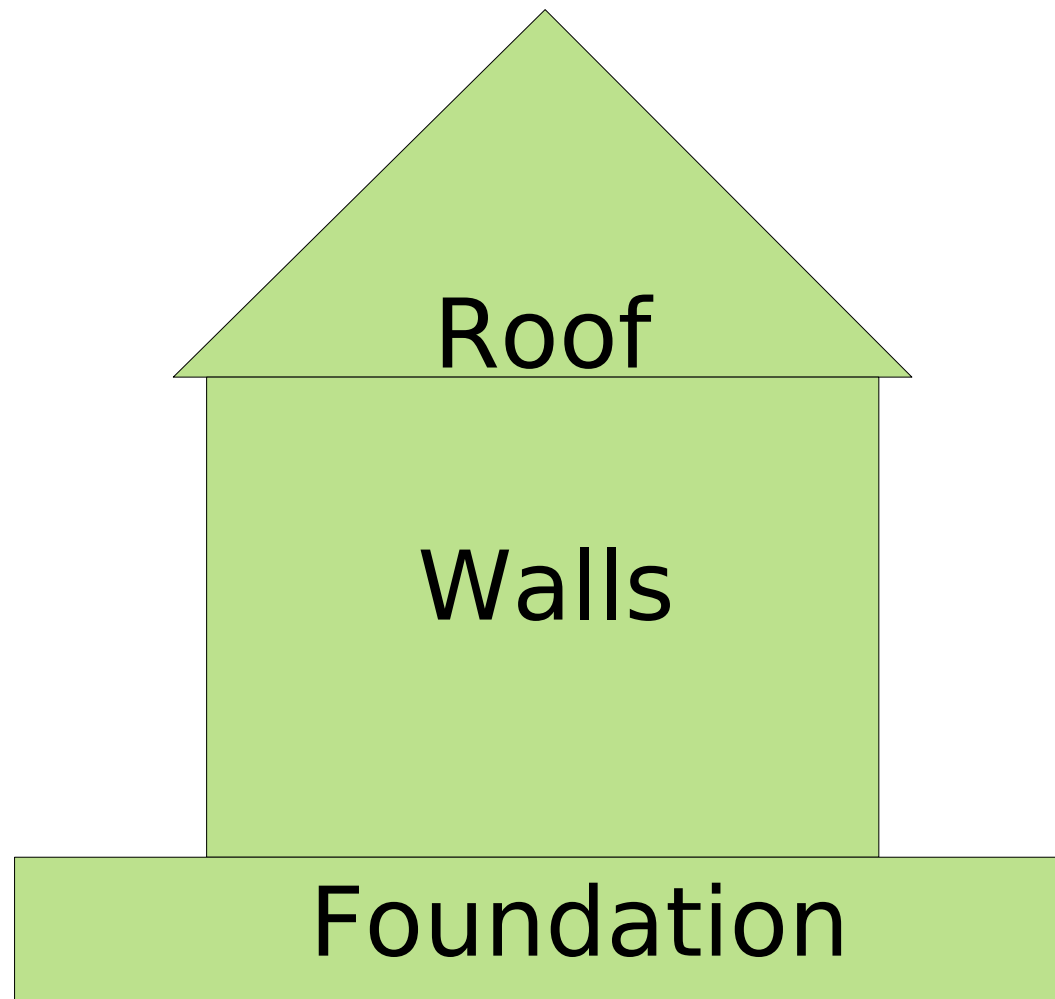
Distribution



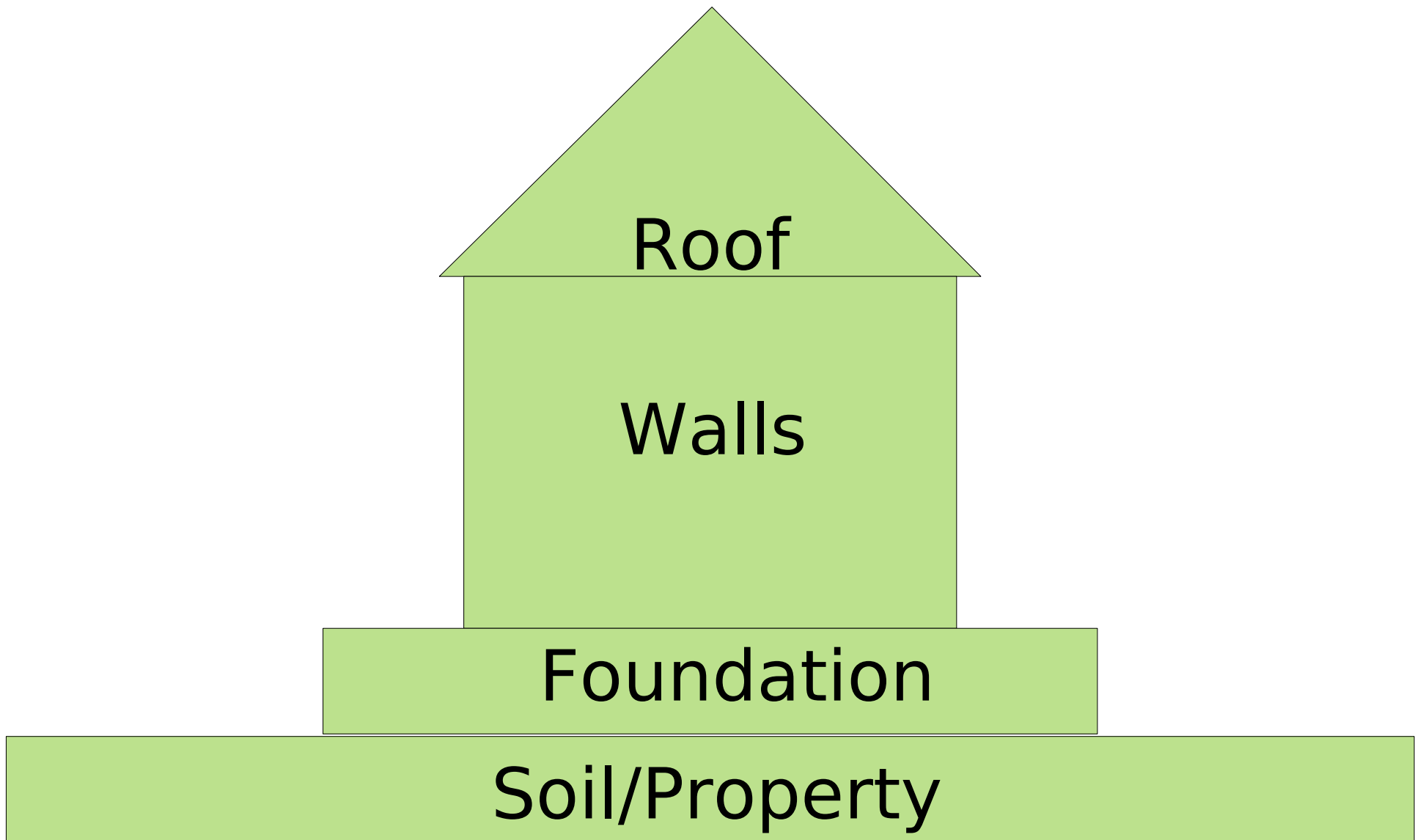
Distribution





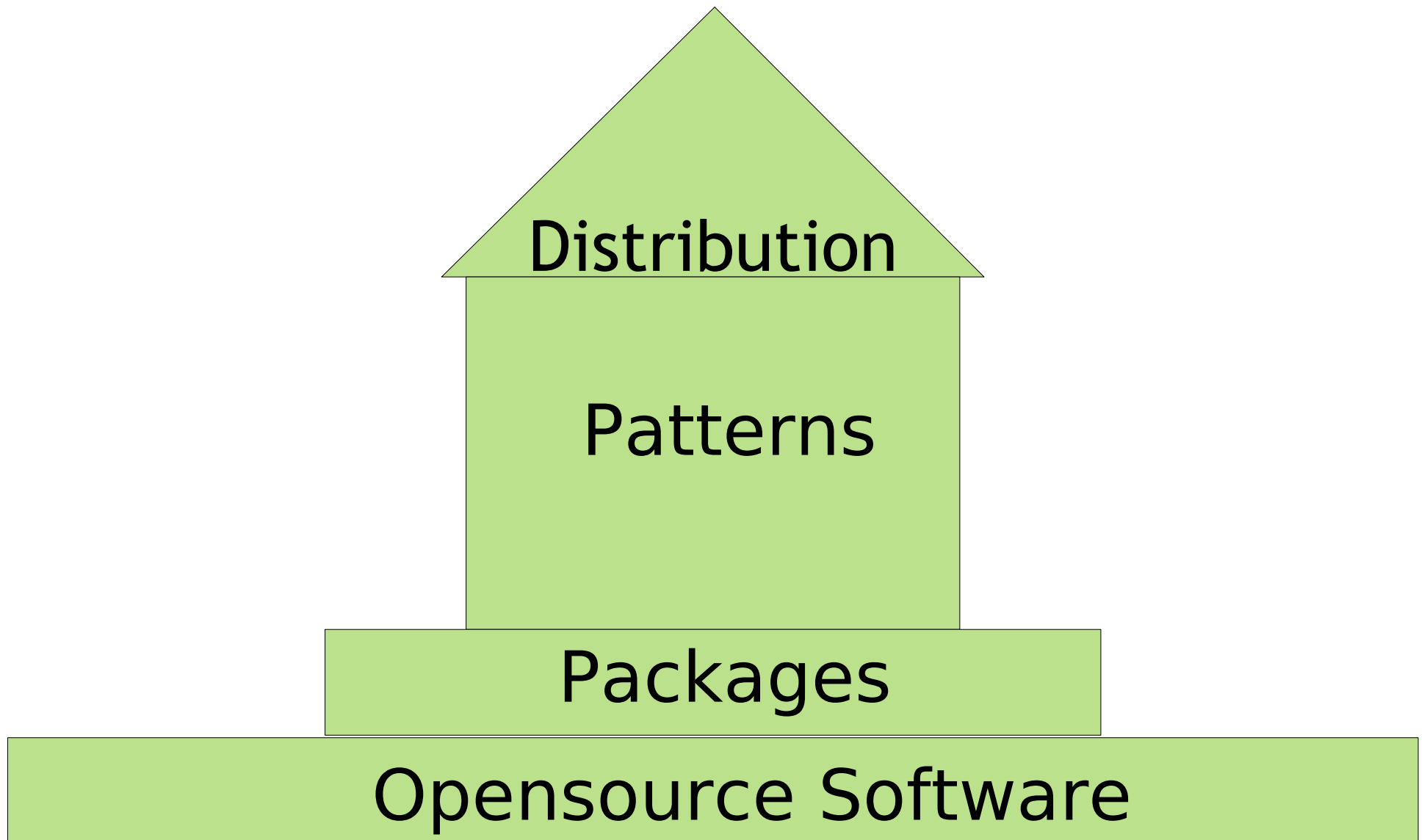


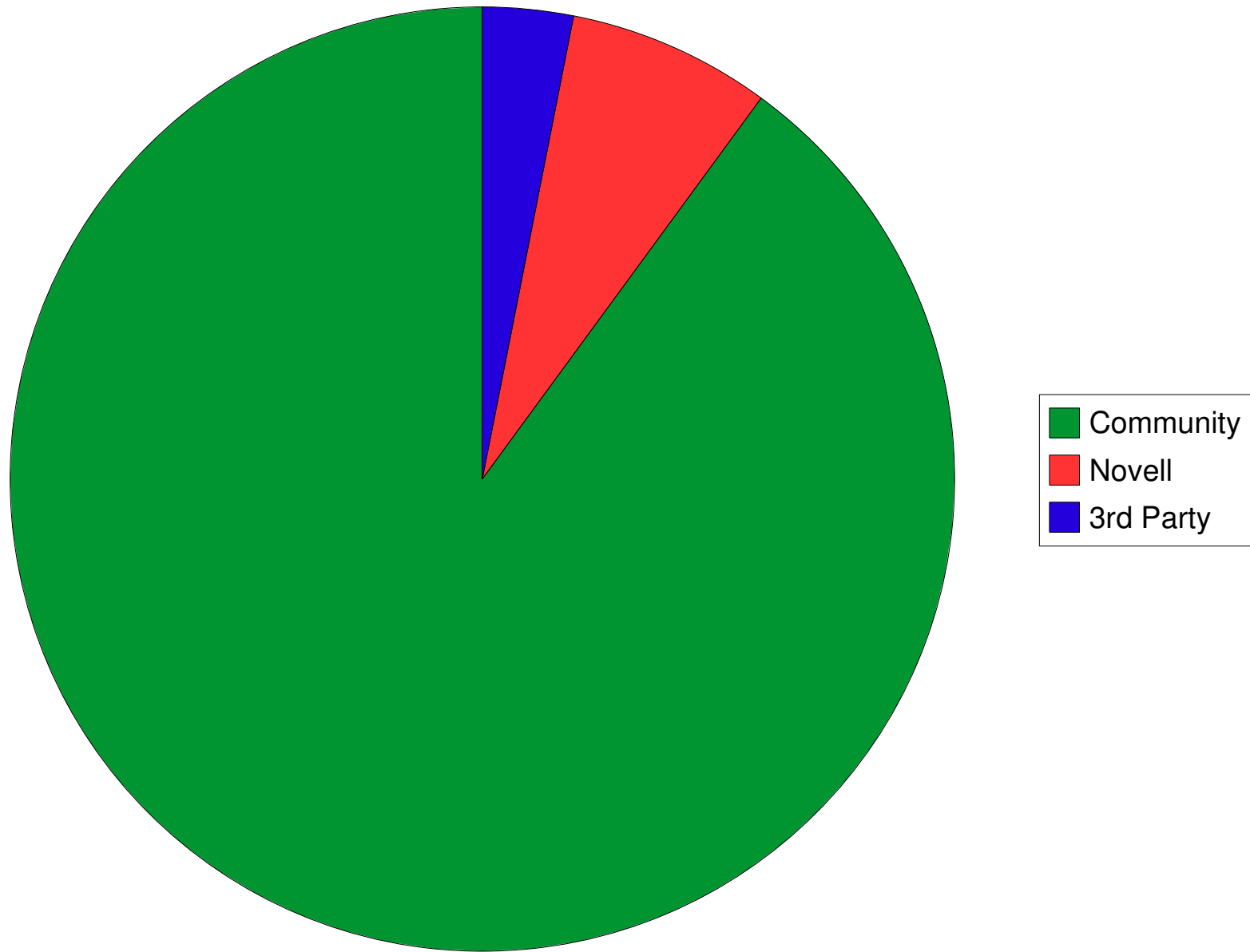


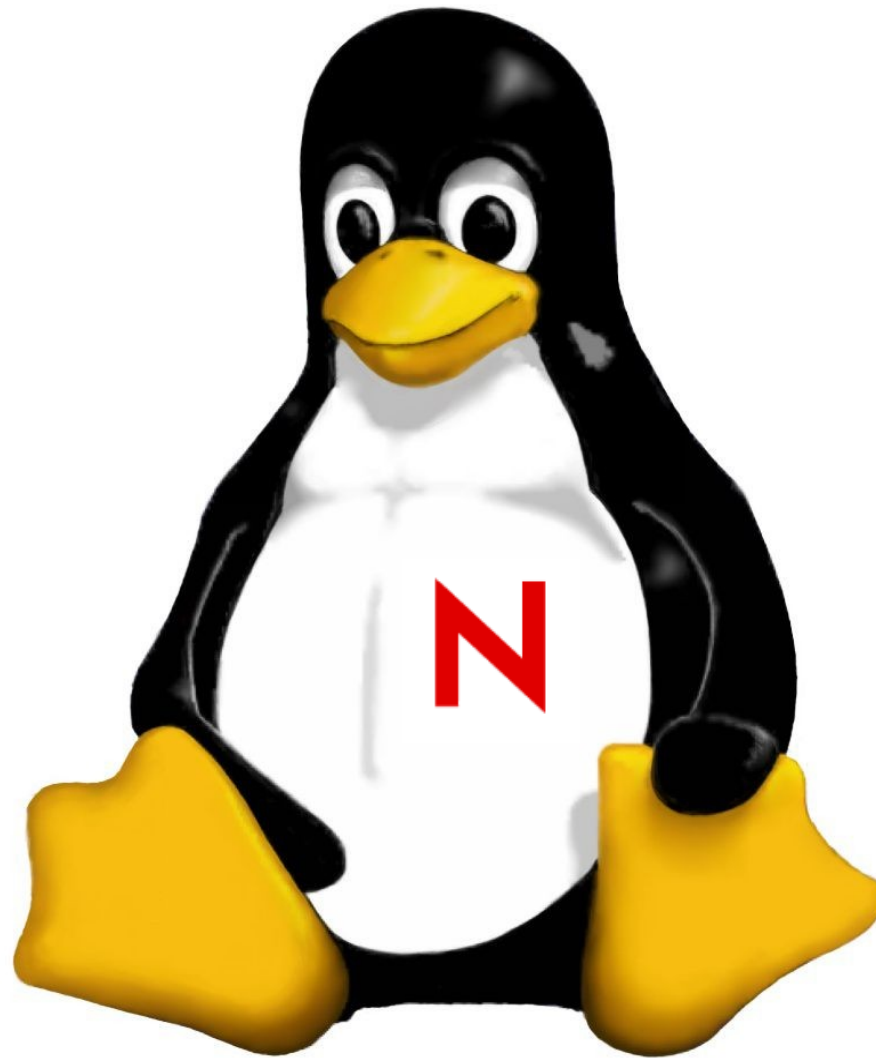


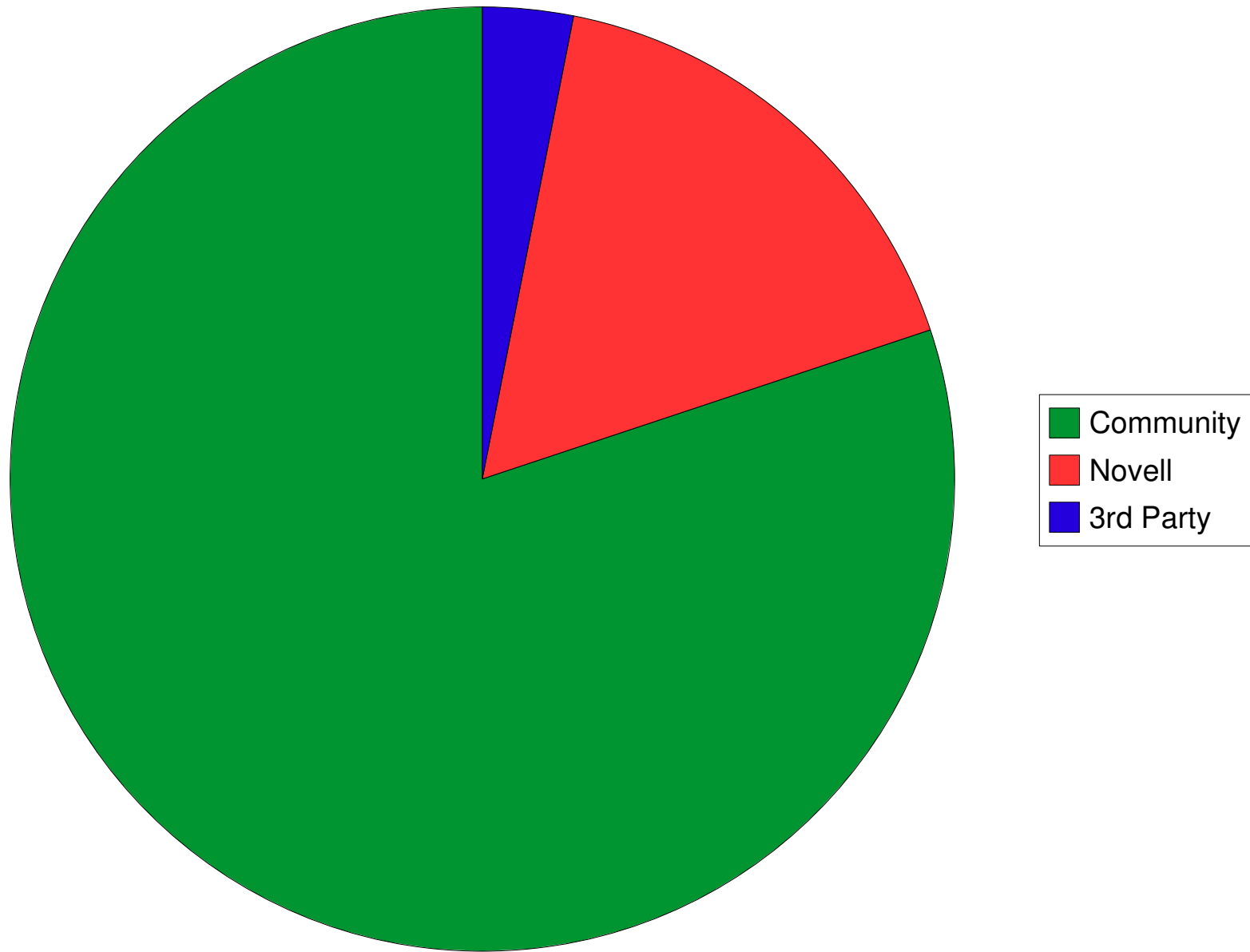


Distribution







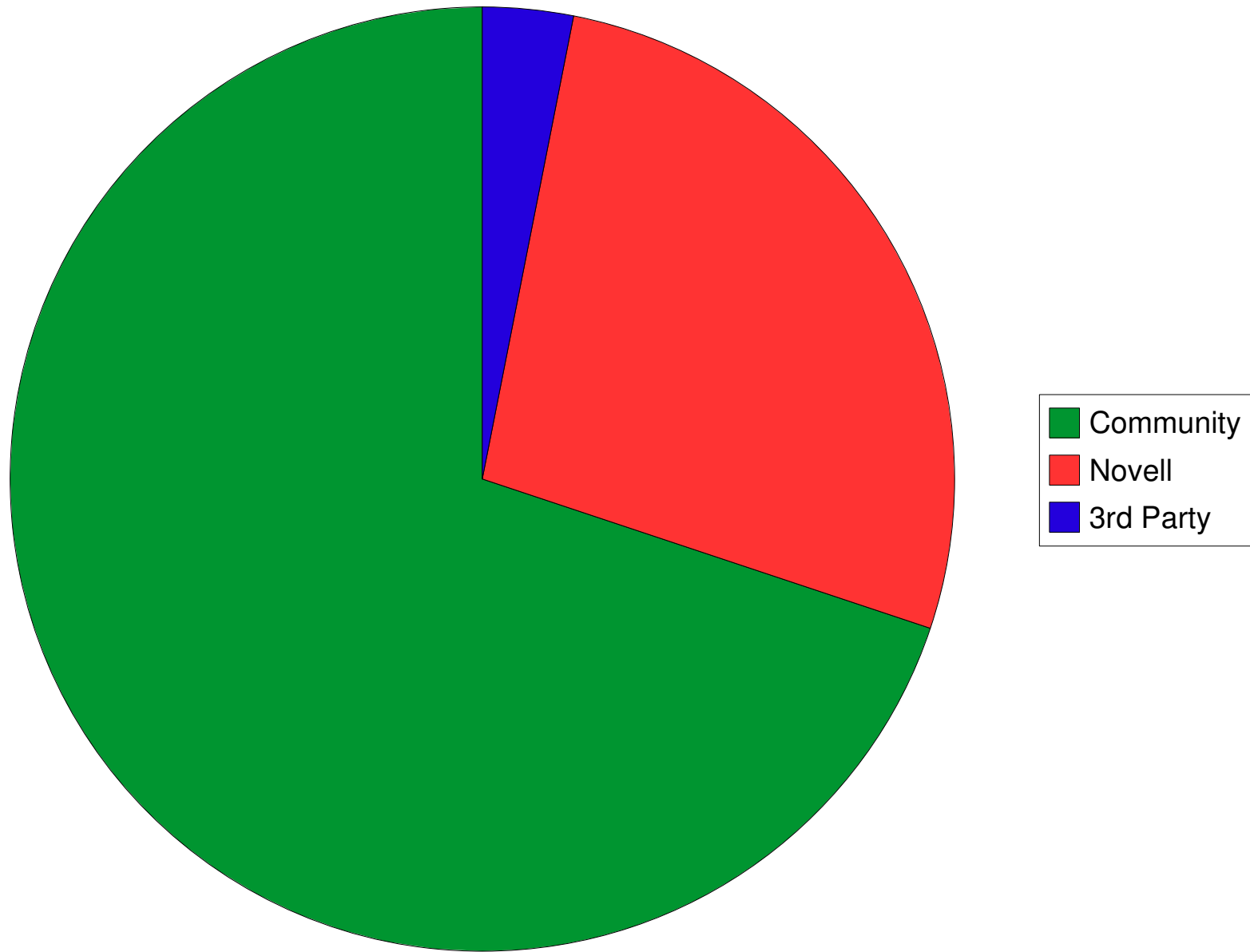


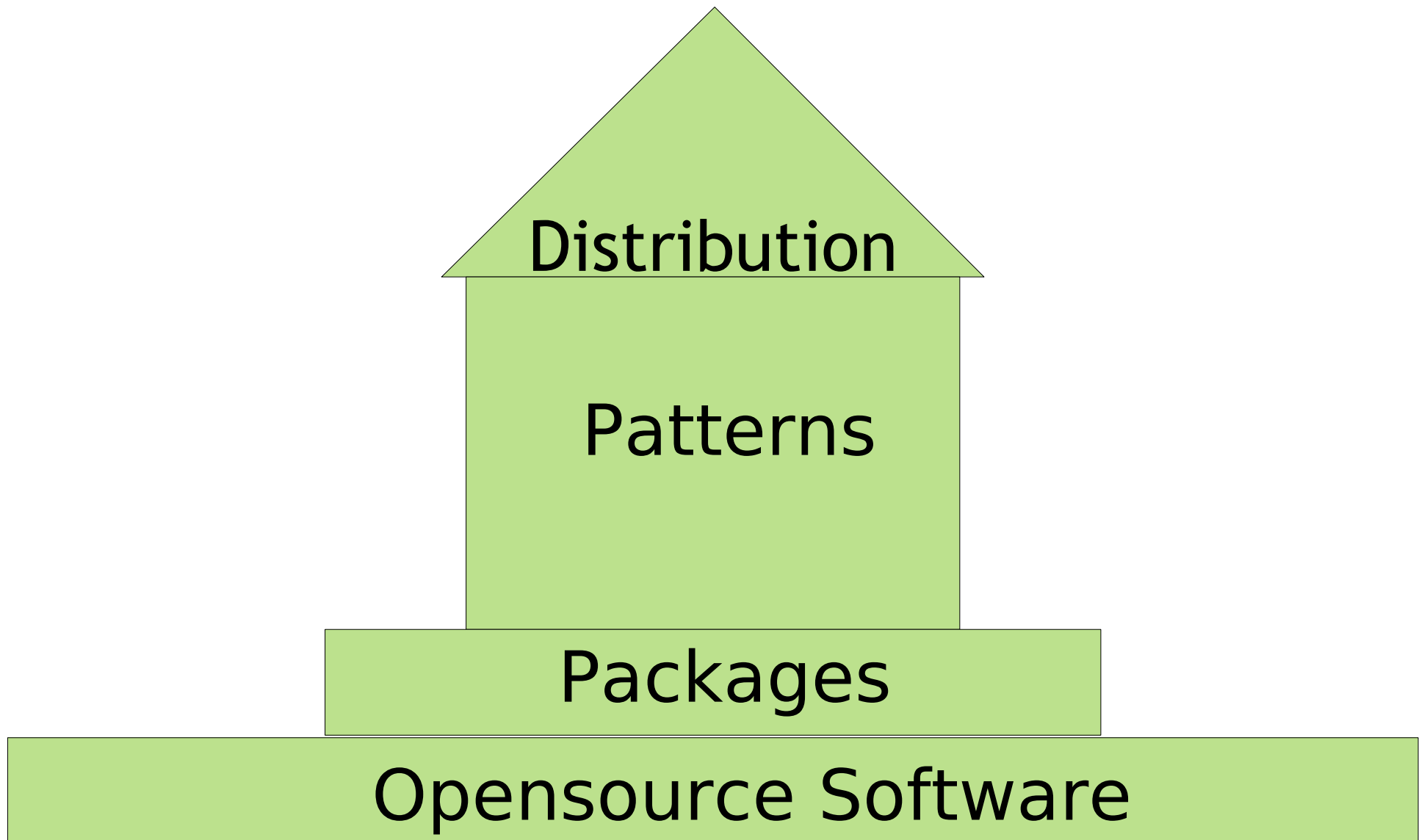
Sorry, but  
Nobody is perfect!

- 
- 3100 software projects
  - 7200 patches

~2,3 Patches







# Development Model

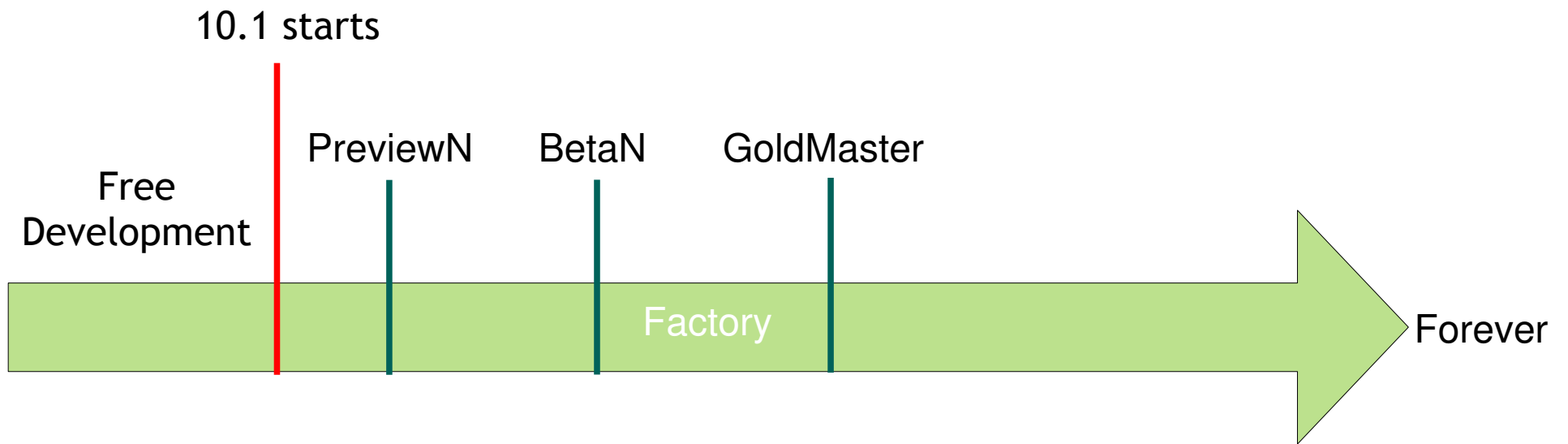
Free  
Development

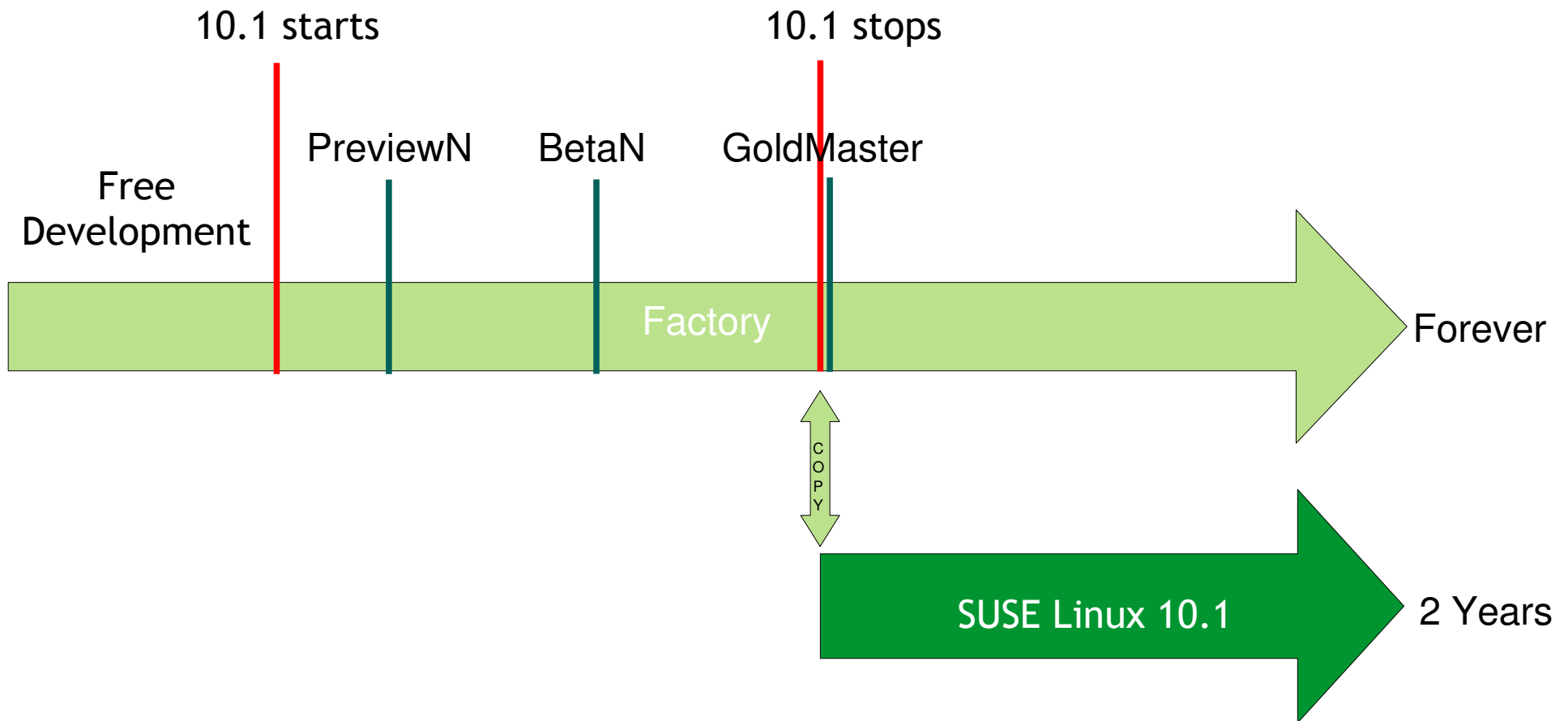
Factory

Forever

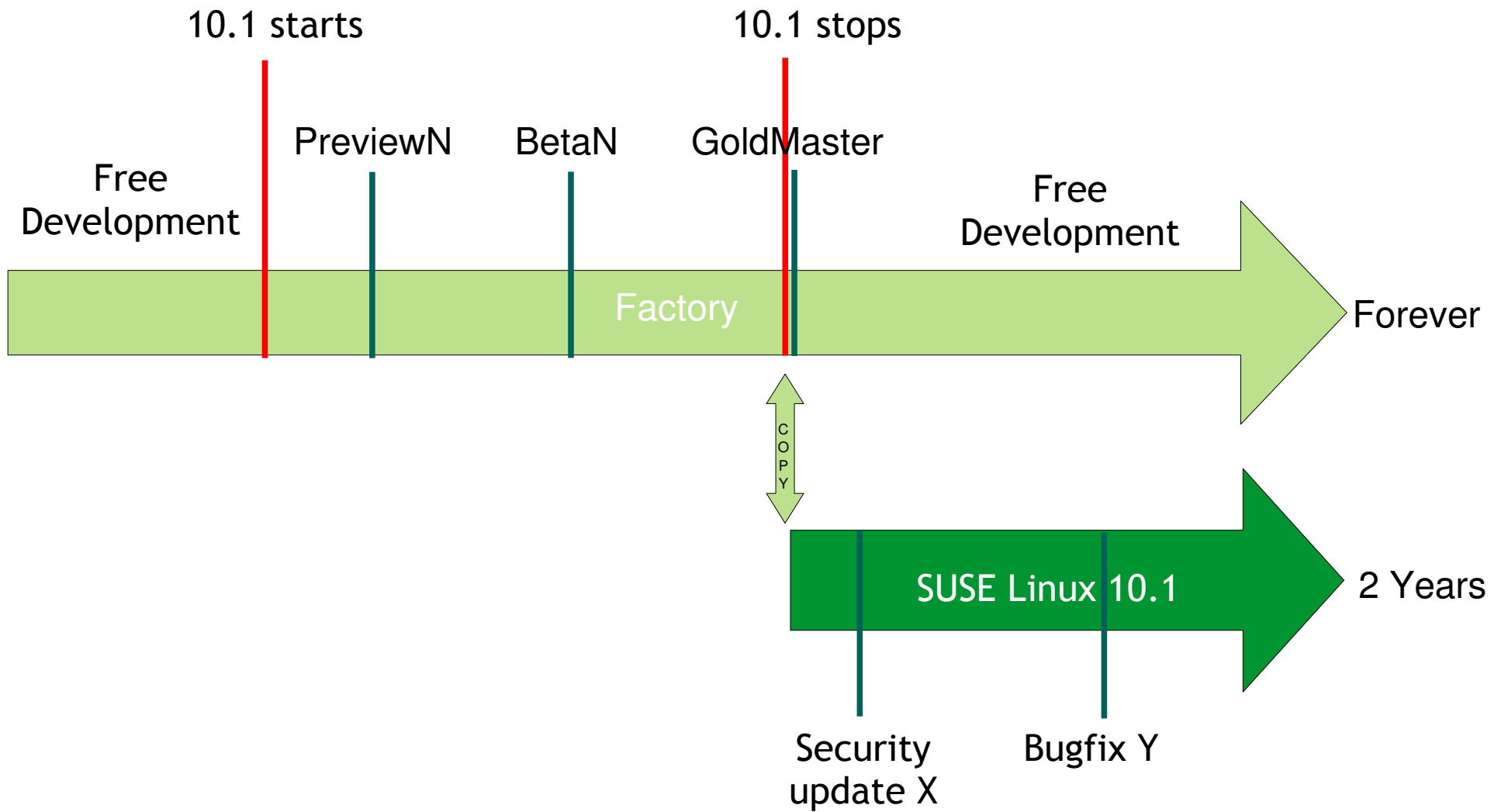












Packaging

Sources

# Programming Languages

- CVS / SVN

---

- **CVS / SVN**

- **CVS Book**

- <http://cvsbook.red-bean.com/>

- **CVS Reference Card**

- <http://tnerual.eriogerg.free.fr/cvsqrc.pdf>

- **SVN Book**

- <http://svnbook.org>

- **SVN Reference Card**

- <http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf>

- 
- CVS / SVN
  - GNU make

- GNU make

<http://www.gnu.org/software/make/manual/>



- 
- CVS / SVN
  - GNU make
  - configure / auto tools

---

- configure / auto tools

<http://sources.redhat.com/autobook/>

<http://www.suse.de/~sh/automake/>

- 
- CVS / SVN
  - GNU make
  - configure / auto tools
  - patch / diff

---

- patch / diff

<http://www.gnu.org/software/diffutils/manual/diff.html>

- 
- CVS / SVN
  - GNU make
  - configure / auto tools
  - patch / diff
  - quilt

- quilt

<http://www.suse.de/~agruen/quilt.pdf>

RPM

- rpm
- rpmbuild



---

## • General RPM Documentation

- <http://www.rpm.org/max-rpm/>
- <http://fedora.redhat.com/docs/drafts/rpm-guide-en/>
- <http://linux01.gwdg.de/~pbleser/files/presentations/rpm-packaging.pdf>
- <http://www.gurulabs.com/goodies/guru+guides.php>

---

- SUSE Package Conventions

[http://forge.novell.com/modules/xfref\\_library/detail.php?reference\\_id=1544](http://forge.novell.com/modules/xfref_library/detail.php?reference_id=1544)

Build Tools

- y2pmbuild
- build

---

- **y2pmbuild**

[http://en.opensuse.org/SUSE\\_Build\\_Tutorial](http://en.opensuse.org/SUSE_Build_Tutorial)

- **build**

<http://www.novell.com/cool solutions/feature/11793.html>



- openSUSE Build Service

[http://en.opensuse.org/Build\\_Service\\_Team](http://en.opensuse.org/Build_Service_Team)

Maintenance

- Security
- Bugfixes
- Feature Updates



Questions?

Novell.®

## General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of Novell, Inc. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.



Novell®

# Packages - The bricks we build with

Hendrik Vogelsang  
Packager  
Novell



Novell.

hi there

my name is hendrik vogelsang

thank for coming here this early on a sunday morning to listen to my talk. i really appreciate it.

Some words about me. Im with suse since early 2000. Ive started in the installation support. Supporting SUSE Linux customers. Then in 2003 i took over the coordination of everything development related in the installation support. Like bugreporting, developer support for our supporters, the beta test of the installation support team and things like that. Since a little over one year i work in the development department as a packager. I maintain somewhere around 100 packages. Among them are core services like the ftp servers we have, the ppp daemon and gnutls. I would like to tell you today about packaging



## The purpose of this talk

In the end i would like all of you

- \* What a distribution is from the packagers point of view.
- \* how we develop distributions.
- \* and why packaging is the very core of it.

This talks is not another rpm manual and its certainly no replacement for packaging experience.



Sources

Okay lets start at the very beginning. Source.

The typical output of an opensource project is the following:



project.c, project.h

configure, Makefile

man page, README

The projects source source code.

Tools which control the generation of executables from the program's source code.

Documentation

The usual workflow to get the project running is



---

prepare the source code

build executables

install executables  
and documentation

Preparing the the source code for the build. That can be running some kind of configure script, configuring settings with an editor directly in some file, checking for other software that this project depends on etc.

Building the executables with the build tools of the project.

Install the executables and documentation into the system.

Now you might ask: Why is that not enough?





Packages

Why do we need packages?



150.000

A typical linux desktop system consists of round about 150.000 files. The average software project installs somewhere around 150-200 new files. The root directory alone holds usually 18 subdirectorys. You get the idea that we are dealing with a lot of files here.

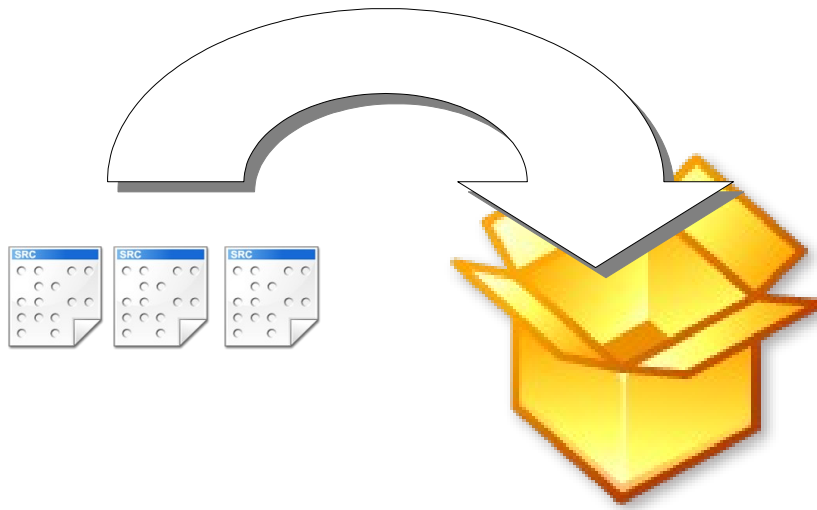
Especially installing and uninstalling software means shoveling files around.

And its not only about moving files but for instance you need to make sure that software X does not overwrite files of software Y and render it useless. You don't want files of software X left on your system if you uninstall it and so on.

To summarize: You need to manage files somehow.

Ok we need to manage files.

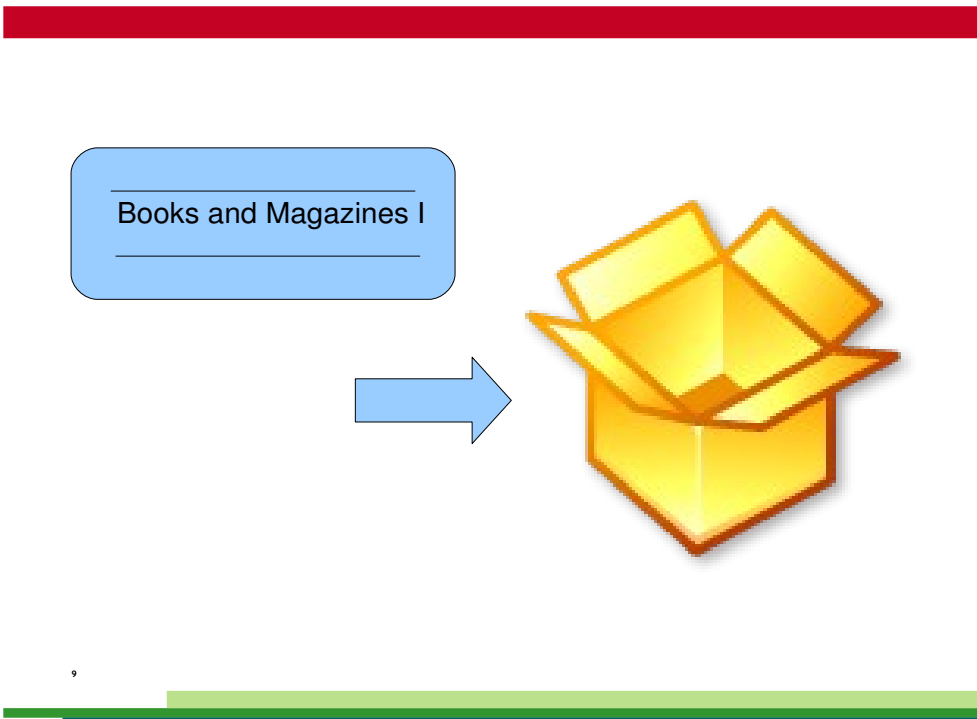
So how do you manage files?



8

Simple. You put the files into a box.

Then, because you want to know what's inside the box/container without unpacking it, you put a label onto it.



That scales for exactly one box. But having one box with 150.000 files does not solve anything. You need to have more boxes. And as you get more boxes you you better start doing this

#### Package Manifest



#### Books and Magazines I

- Foundation by I. Asimov
- Cryptonomicon by N. Stephenson
- Wired (all of 2005)
- Time (all of 2005)

Packed 2006.03.01 by hvoegel



#### Books and Magazines II

- HHGG by Douglas Adams
- 1984 by George Orwell
- Wired (all of 2004)
- People (all of 2004)

Packed 2005.12.06 by hvoegel

You better start writing down what the box with label X contains.



- Files (Box)
- Meta Data (Label)
- Package Database (Manifest)

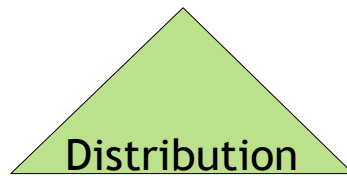
And that is exactly what the package system that we use does.

An RPM package consists of files and meta data and the RPM package manager keeps track of this data.



Distribution

So we know we have source which we need to stuff into packages. What else is needed for a Distribution?

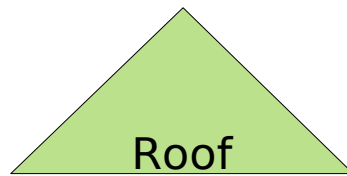


13

The usual look at a distribution is this.

That is as if you say that a house is something like this:

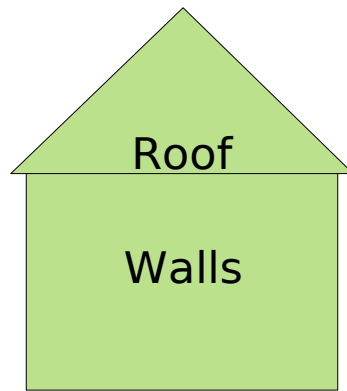




14



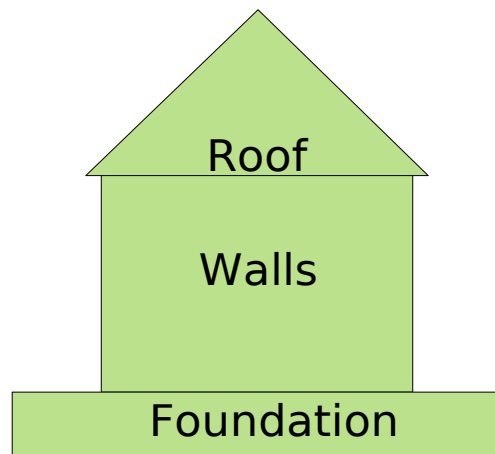
But we all know that you actually need a little bit more



15

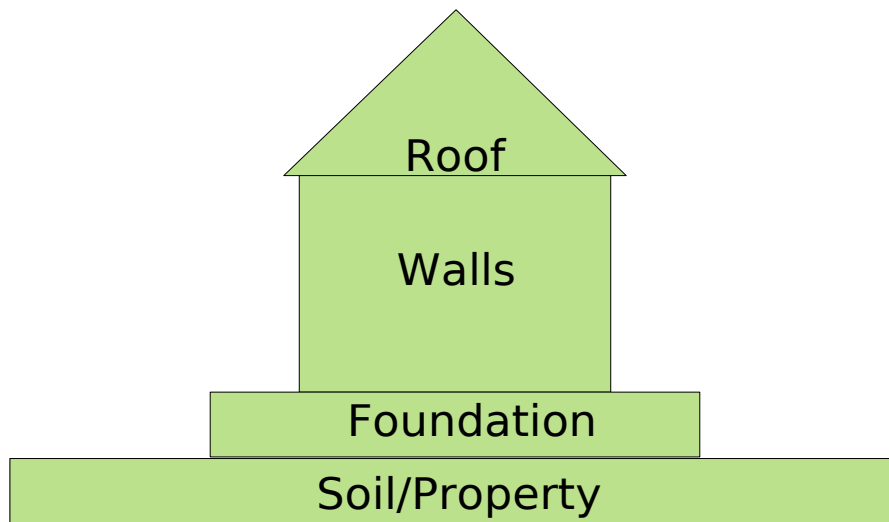


Like walls



16

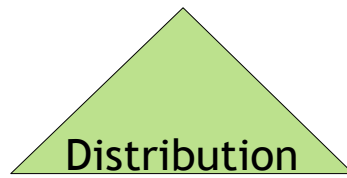
a good strong foundation



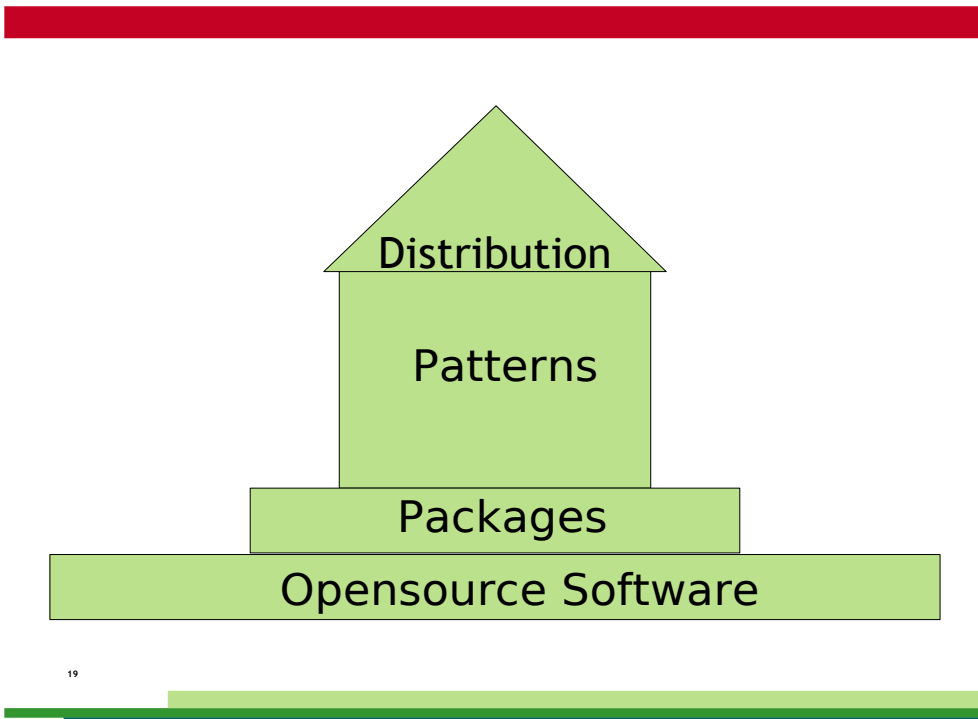
17



and some beautiful piece of land to build it on.



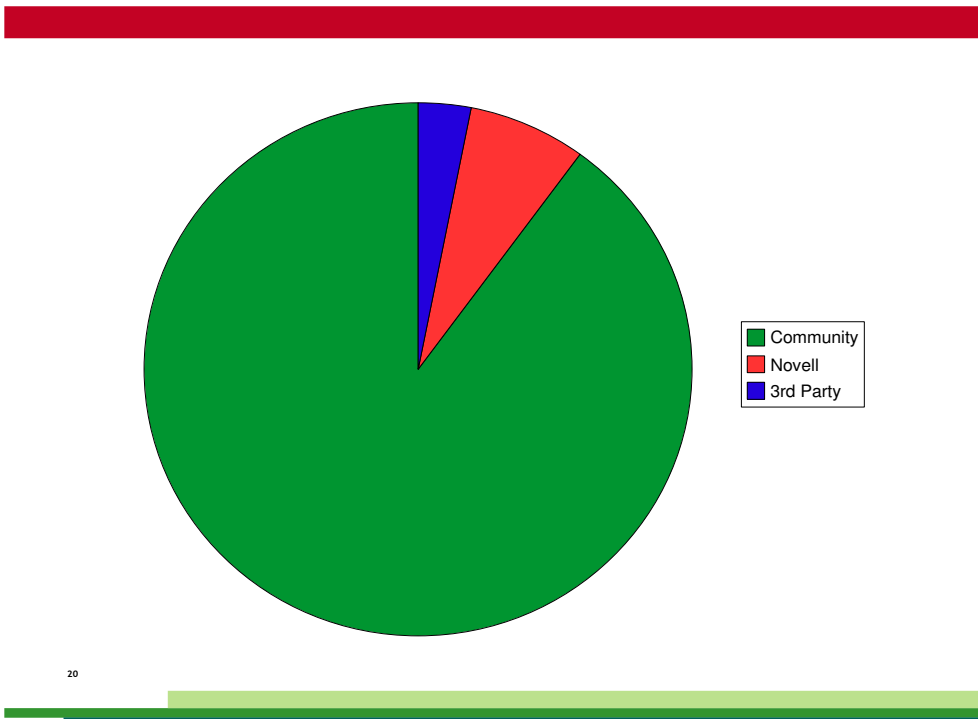
So a distribution isnt this



Its this!

A distribution are usage patterns that include packages that include opensource software.

Usage patterns are just definitions of tasks that one can do. A usage pattern can be for instance “serve webpages trough a webserver”. This pattern then includes all packages that you would need for that.



Some word on where the source code comes from.

This is a chart shows where the software in our distributions comes from.

There is a huge green chunk of software that represents the code from the opensource community

An smaller red chunk of software that Novell develops in-house. Like the YaST system administration tool.

And an even smaller blue chunk of 3<sup>rd</sup> party software that is developed by a partner of Novell.

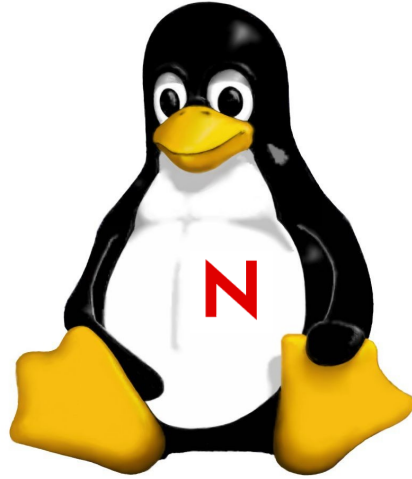
Looking at this chart one imediately asks:

So what does Novell do?

There is only this small chunk of the cake?

That cant be all, can it?

OF COURSE NOT

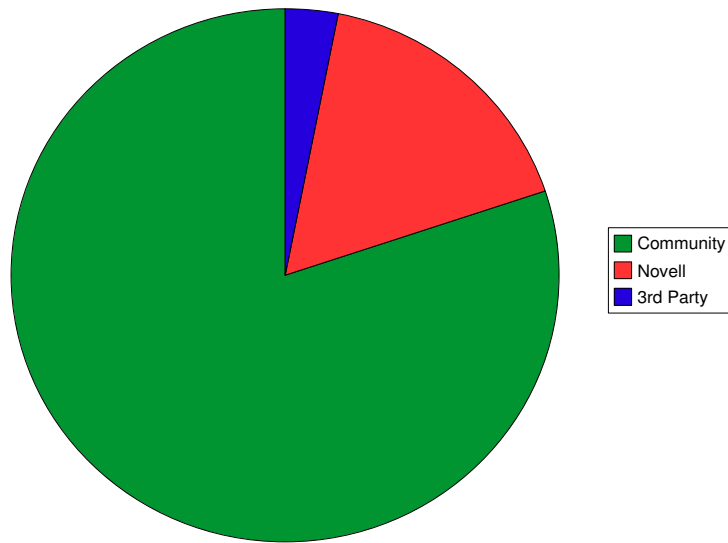


21

First: Novell is a big part of the community.

- \* We employ more than 20 kernel developers
- \* We employ more than 10 toolchain developers
- \* We employ more than 30 desktop developers
- \* We employ more than 50 lead developers of individual community projects.






22



So the reality looks more like this. The red chunk got a good amount bigger.



Sorry, but  
Nobody is perfect!

23



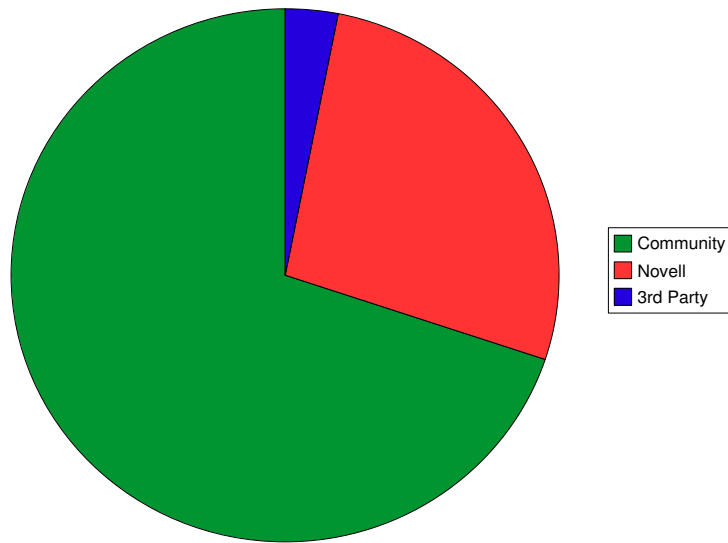
The second reason is sad but true: Nobody is perfect. Not even we, the opensource community.

Software needs to be fixed to compile, integrate or even run correctly.



- 3100 software projects
  - 7200 patches
- ~2,3 Patches

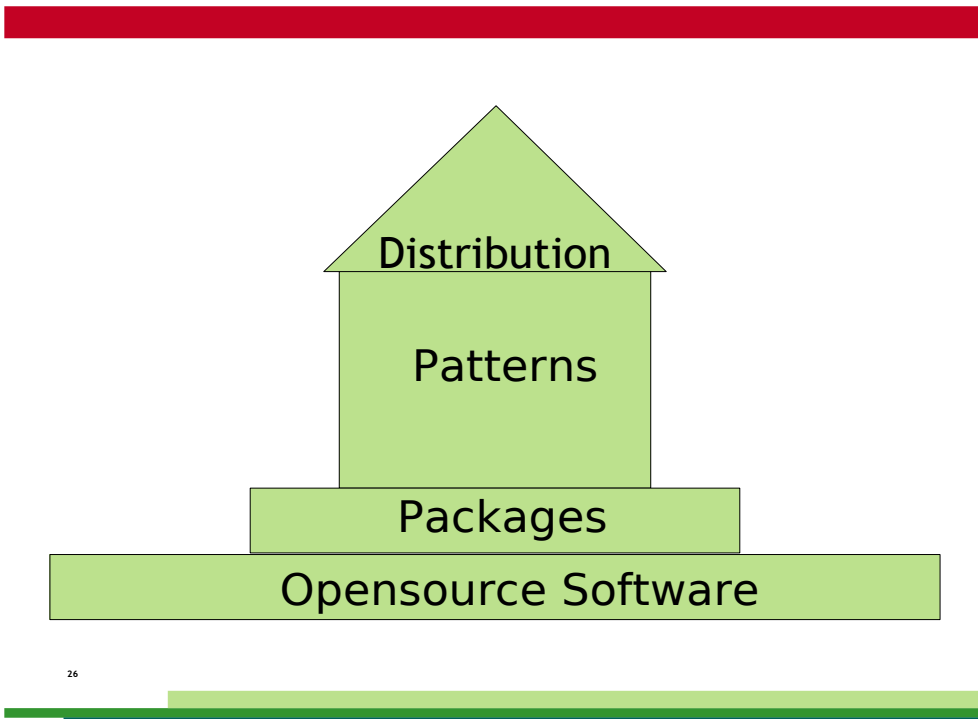
Some numbers to back that outrageous claim up.  
At the moment we have somewhere around  
3100 software projects in our build base.  
And round about 7200 patches.  
That means the average number of patches for an  
software project is ~2,3.



25



So this is the reality. Novell touches a big chunk of all the code that is inside a distribution.



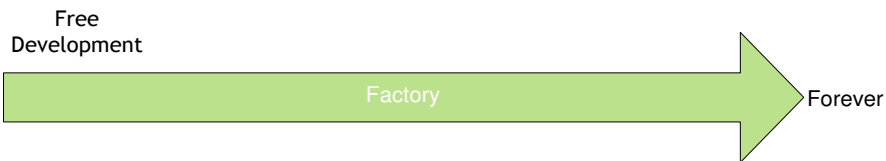
So this is how we do it.

We take opensource software, review/integrate it and because everything is a file we need to package the files to keep them manageable. We need to define some use patterns to group packages and once we did that we bundle usage patterns and call all that a distribution.



## Development Model

So now that we actually know what a distribution is the next thing i would like you to understand is how we deal with distributions.



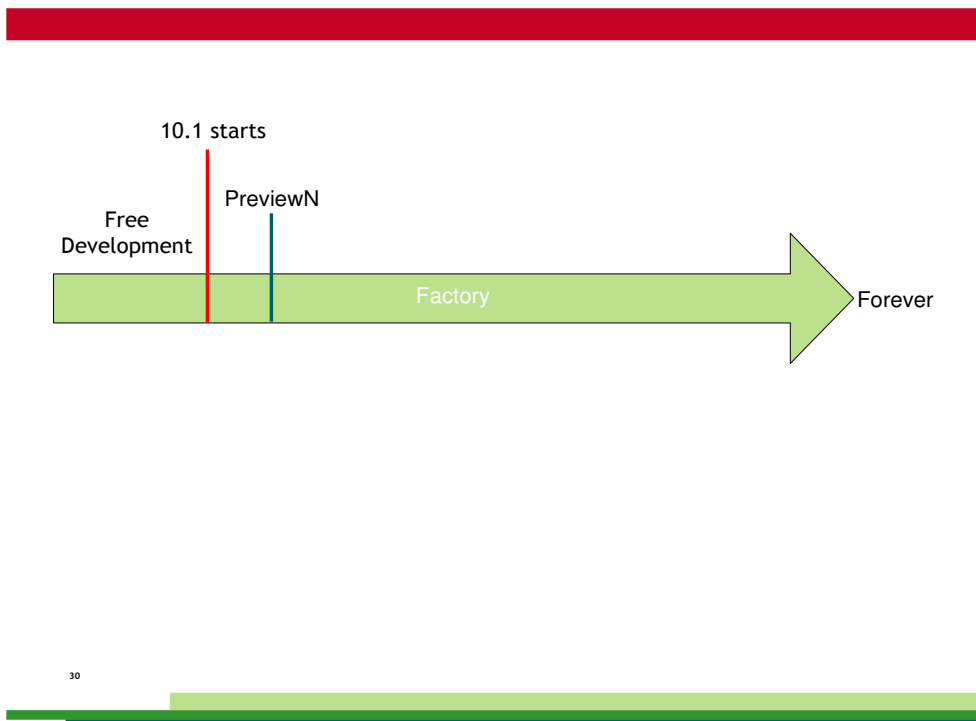
We have a "code stream" that is constantly developed. It is called Factory. Every time a packager submits a fix, version update, new feature or new package it is built in Factory.



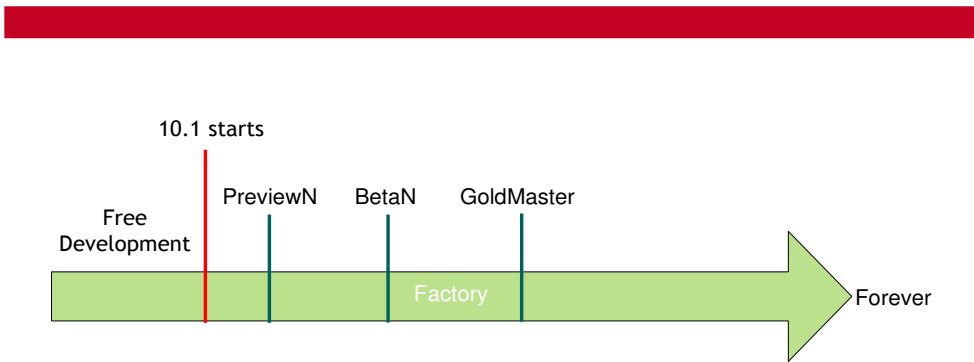
29

Then from time to time we "freeze" the Factory stream and allow only fixes to go in. No new features, no version updates and no new packages.

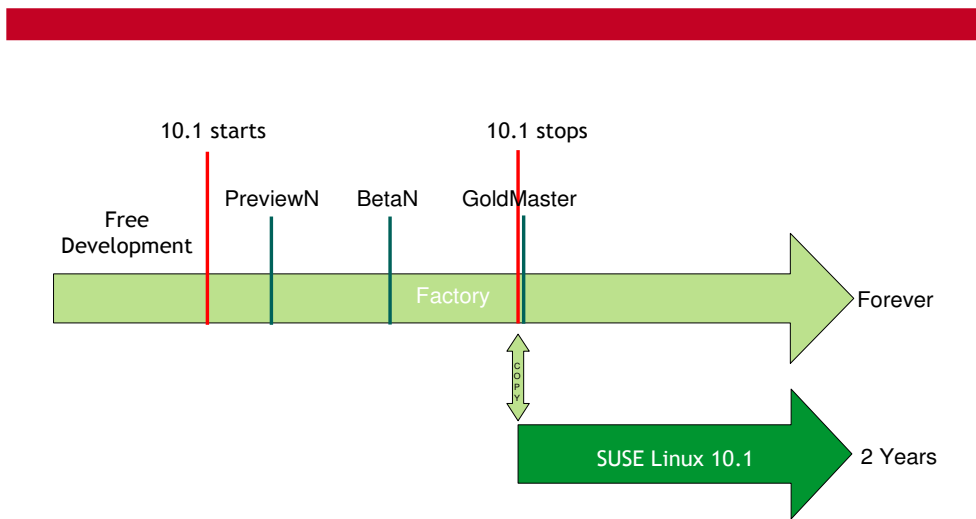




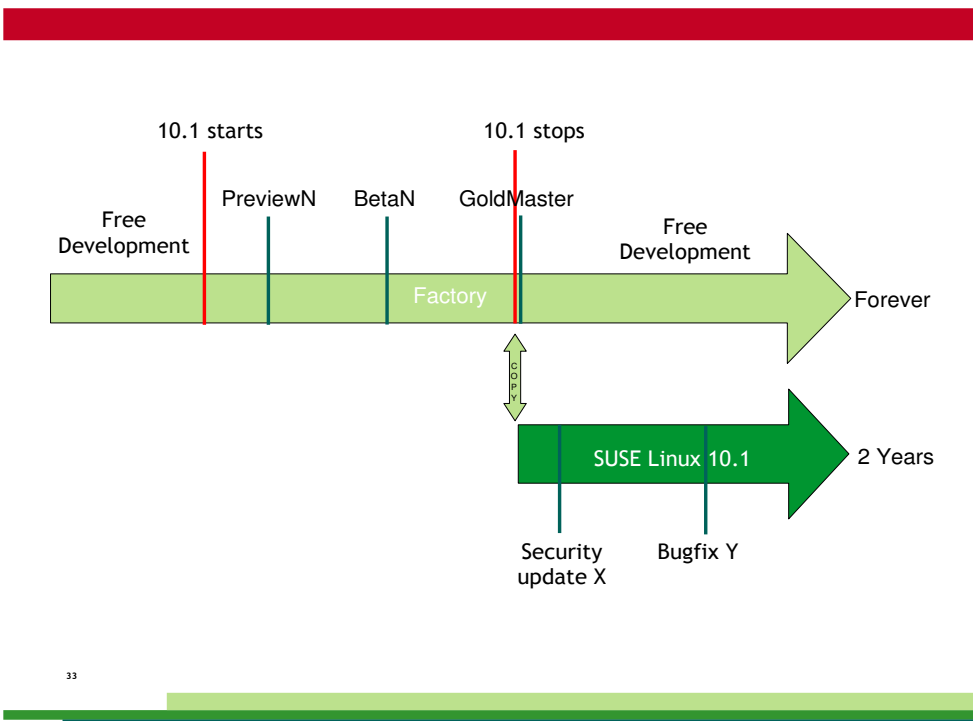
Then on fixed dates we build installable media sets from the Factory stream so that everybody can test the same "state" of the "frozen" Factory stream. First they are called previews



Previews lead to betas which lead to release candidates (RC), which finally lead to a gold master (GM).



Once we have a goldmaster we copy the frozen stream to another location and unfreeze the Factory stream again.



Then you have two streams:

Factory that receives:

- \* fixes
- \* new packages
- \* new features
- \* new versions of packages

A released product like SUSE Linux 10.1 that receives:

- \* Security fixes
- \* fixes for serious bugs



## Packaging

That was a little high level overview on what we are dealing with when we talk about packaging.

So now for details. But keep in mind that i wont make this another RPM manual. I cant teach you to become a packager in one hour. So i wont pretend i can!

What i can do is tell you what tasks packaging involves and point you to resources where you can find in depth information about the tasks.



## Sources

As we learned you need to be able to handle source code. Don't be afraid! Most opensource projects are really straight forward to build. And if you have problems with that most of them have really helpful developers. But nevertheless you should make yourself familiar with the following things

## Programming Languages

Ok heres the deal. If you package some software written in language foo you can either learn foo and fix stuff yourself or you can depend on the developers of the project to help you.

Both ways are fine but learning the language the project in your package is written in has of course advantages. A: You are fixing stuff faster and B: you help the project.

- CVS / SVN

Version control systems. You actually need mostly the “read” (checkout) part of those.

A usual task for a packager is to check if bugs are fixed in the development version of some project.

Also extracting specific changes to files included in a project happens often.

For some projects there even might be no stable version yet and you need to check out the project from a version control system.





## • CVS / SVN

- **CVS Book**

<http://cvsbook.red-bean.com/>

- **CVS Reference Card**

<http://tnerual.eriogerg.free.fr/cvsqrc.pdf>

- **SVN Book**

<http://svnbook.org>

- **SVN Reference Card**

<http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf>

CVS and SVN are both very well documented.

For CVS there is the book “Open Source Development with CVS” you can read it online or order a copy at O'Reilly. Also there exist a lot of reference cards on the net.

SVN documentation is mainly in the book “Version Control with Subversion”. You can read that one online too or order your copy from O'Reilly. Of course for SVN there are also reference cards.

- CVS / SVN
- GNU make

Source code build systems. For instance GNU make.

Make is used in a lot of projects to control source code compilation. As source code compilation is also at the heart of packaging every packager gets in contact with GNU make very often.

Of course there are other tools like perl Makefiles, python install scripts, iMake, scons and so on. But usually you can tackle those once you've understood the concept of make.

- GNU make

<http://www.gnu.org/software/make/manual/>

There is a nice make manual that covers both tasks a packager has to do. Fiddling with Makefiles and invoking make. You can read it online at the GNU homepage or order a printed copy from the FSF.

- CVS / SVN
- GNU make
- configure / auto tools

configure and auto tools are a set of tools to make source code builds portable to various open systems. configure checks the system you build on for various things and generates Makefiles that match your system. The auto tools are another level above and generate configure scripts and Makefiles for configure scripts to process. This is a very complex topic and with lots of layers of redirection. Mostly it “just works”. If not its good for a packager to know whats failing. Fixing it is a different story...



- configure / auto tools

<http://sources.redhat.com/autobook/>

<http://www.suse.de/~sh/automake/>

Because configure scripts mostly are shell scripts you should be familiar with the bourne again shell (bash).

There is a comprehensive book about the auto tools called “GNU Autoconf, Automake and Libtool” you can read it online or order a printed copy at Sams publishing.

Stefan Hundhammer made a really nice presentation that is an overview of all the source code build systems. From simple Makefiles to configure scripts and the auto-tools. You can read it online at his suse user directory.

- CVS / SVN
- GNU make
- configure / auto tools
- patch / diff

RPM has the concept of pristine sources. So you if you want to change something in the source files you need to generate patches. You can use the GNU diffutils for that.



- patch / diff

<http://www.gnu.org/software/diffutils/manual/diff.html>

The GNU diffutils are explained in detail on  
[gnu.org](http://gnu.org)

- CVS / SVN
- GNU make
- configure / auto tools
- patch / diff
- quilt

Because we change the source with patches we might run into problems handling all those patches. Some might depend on each other and therefor have to be applied in a certain order. With one to ten patches a packager can usually handle this on it own. But especially with large projects you might end up with ten or more patches with interdependencies and stuff like that. If you ever come into this situation then you might invest some time to learn how quilt works. Its a patch management tool that helps keeping track of many patches.





- quilt

<http://www.suse.de/~agruen/quilt.pdf>

Quilt originally was based on Andrew Morton's patch scripts. You can find documentation online at Andreas Grünbachers suse home directory.



RPM

I think everybody should see now that packaging is a very complex task. Alone for handling the source code you need experience with a lot of tools and even coding experience is of great help.

Handling source code is a big part of packaging the other big part is handling the RPM package manager.

- rpm
- rpmbuild

The RPM package manager is a fairly straight forward project. It involves three things. rpmbuild to build packages. The rpm commandline tool to operate it and the rpm database. A packager should know about all three to some extent. Especially rpm and rpmbuild and the control files for rpmbuild, called specfiles, are of great interest.



## • General RPM Documentation

- <http://www.rpm.org/max-rpm/>
- <http://fedora.redhat.com/docs/drafts/rpm-guide-en/>
- <http://linux01.gwdg.de/~pbleser/files/presentations/rpm-packaging.pdf>
- <http://www.gurulabs.com/goodies/guru+guides.php>

There exists a lot of Documentation on the net about RPM. rpm.org is a good starting point. There is a online book about RPM called “Maximum RPM”. Fedora has a nice RPM guide too that seems to be based on Maximum RPM.



- SUSE Package Conventions

[http://forge.novell.com/modules/xfref\\_library/detail.php?reference\\_id=1544](http://forge.novell.com/modules/xfref_library/detail.php?reference_id=1544)

We at openSUSE use some extensions to RPM specfiles. Things like handling of desktop files, restarting services, sysconfig file handling and things like that. Those extensions are explained in the SUSE Package Conventions on [forge.novell.com](http://forge.novell.com)



## Build Tools

Lets say you created your first package of project foo and want to start building it with rpmbuild. You can do that but you would use your current installation as build environment then. So if project foo depends on project bar you would have to install bar in your current installation. You usually don't want to do that. You want to have a separate build environment to keep your working installation clean and make the build reproduceable. You could just use a second machine or you install everything you need to a separate location on your system and use tools like chroot to operat in in there. Now doing this for 1 build environment is no problem its just like maintaining a second installation including keeping it up to date with security updates, latest packages and so on. But who wants to do that? Thats why there are tools around that help you with that. Build Tools

- y2pmbuild
- build

At the moment you have 2 options. y2pmbuild which is a script around the YaST2 Packagemanager shell or build which is stripped down version of what we use at novell. Both function very similar. They take RPM packages from installation sources and setup the build environment on the fly. Then they run rpmbuild in that build environment.



- **y2pmbuild**

[http://en.opensuse.org/SUSE\\_Build\\_Tutorial](http://en.opensuse.org/SUSE_Build_Tutorial)

- **build**

<http://www.novell.com/cool solutions/feature/11793.html>

y2pmbuild is documented in the SUSE Build Tutorial on the wiki and build is covered in a cool solutions article on novell.com





- openSUSE Build Service

[http://en.opensuse.org/Build\\_Service\\_Team](http://en.opensuse.org/Build_Service_Team)

Of course as you heard yesterday a crucial part of the openSUSE project is the Build Service so i wont go into details now. You find informations about it on the wiki.



Maintenance

Now comes the hard part! Spitting out RPM packages, even in a good quality, is easy compared to maintenance of packages!

- Security
- Bugfixes
- Feature Updates

A good packager cares for his packages. He follows the projects he packages, keeps track of security problems, bugfixes and new versions. He tries to help people that have problems with his package or even with the included software.

This is the task where most packagers fail. You find tons and tons of RPM packages on the net. The problem is to find reliable, well maintained repositories.

You have to have a lot of drive and time to maintain packages. I can tell you from personal experience it eats a lot of time. But if you don't do it your packages are mostly worthless. I'm sorry but that's the truth.



Questions?

What better end is there then telling the truth? ;-)

Thank you all for your patience during the last hour. I hope i was able to give you an overview about what packaging means. Are there any questions?

**Novell®**

#### General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of Novell, Inc. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.



**Novell.**